NAS Technical Report, NAS-05-011, June 2005

Parallelization of Gauss-Seidel Relaxation for Real Gas Flow

Seokkwan Yoon*, Gabriele Jost** and Sherry Chang***

NASA Ames Research Center
Moffett Field, California 94035

Methods using OpenMP directives have been studied to parallelize an inherently sequential Gauss-Seidel algorithm on shared memory computers. Both hyperplane and pipeline parallelization schemes have been implemented to a non-equilibrium real-gas flow simulation code. The effects of different parallelization strategies and grid sizes on the parallel efficiency have been investigated on SGI Origin and Altix systems.

I. Introduction

Exploration of the solar system has revived an interest in computational simulation of chemically reacting flows since planetary exploration vehicles exhibit non-equilibrium phenomena during the atmospheric entry of a planet or a moon as well as the reentry to the Earth. Stability in combustion is essential for chemical propulsion systems.

Numerical solution of real-gas flows often increases computational work by an order-of-magnitude compared to perfect gas flow partly because of the increased complexity of equations to solve. Recently, as part of Project Columbia, NASA has integrated a cluster of interconnected SGI Altix systems to provide a ten-fold increase in current supercomputing capacity that includes an SGI Origin system. Both the new and existing machines are based on cache coherent non-uniform memory access architecture.

The Lower-Upper Symmetric Gauss-Seidel (LU-SGS) relaxation method[1] has been the core solution algorithm for both perfect and real gas flow codes[2-8] including Real-Gas Aerodynamic Simulator (RGAS)[9]. However, the vectorized RGAS code runs inefficiently on cache-based shared-memory machines such as SGI systems. Porting and optimization of the RGAS code to cache based machines have been performed. However, parallelization of a Gauss-Seidel method is nontrivial due to its sequential nature.

The LU-SGS method has been vectorized on hyperplanes in the INS3D-LU code[4] that has been one of the base codes for the NAS Parallel Benchmarks[10]. It is possible to parallelize a Gauss-Seidel method by partitioning the hyperplanes once they are formed. Another way of parallelization is to schedule processors like a pipeline using software[11]. Both hyperplane and pipeline methods have been implemented using OpenMP directives. The present paper reports on the performance of the parallelized RGAS code on SGI Origin 3800 and Altix 3700 systems.

---

* NASA Advanced Supercomputing Division

** Computer Sciences Corporation, Presently at Sun Microsystems

*** Computer Sciences Corporation, Work supported by the NASA Advanced Supercomputing Division under Task Order A61812D (ITOP Contract DTTS59-D-00437/TO #A61812D) with Advanced Management Technology Incorporated (AMTI)

II. Numerical Methods

Let t be time; Q the vector of conserved variables; E, F, and G the convective flux vectors; E_v, F_v, and G_v the flux vectors for the viscous terms. The source term S represents production or destruction of species due to chemical reactions. The three-dimensional Navier-Stokes and species transport equations in generalized curvilinear coordinates (ξ, η, ς) can be written as

$$\partial_t Q + \partial_\xi (E - E_v) + \partial_\eta (F - F_v) + \partial_\xi (G - G_v) = S \quad (1)$$

The governing equations are integrated in time for both steady and unsteady flow calculations. An unfactored implicit scheme can be obtained from a nonlinear implicit scheme by linearizing the flux vectors about the previous time step and dropping terms of second and higher orders.

$$[I + \alpha\Delta t(D_\xi A + D_\eta B + D_\xi C - H)]\Delta Q = RHS \quad (2)$$

where

$$RHS = -\Delta t[D_\xi (E - E_v) + D_\eta (F - F_v) + D_\xi (G - G_v) - S\ ] \quad (3)$$

I is the identity matrix and ΔQ denotes the correction. A, B, C, and H are the Jacobian matrices of the convective flux vectors and the source term respectively. For steady-state solutions, α is set to 1. Artificial dissipation models augment a piecewise-constant cell-centered finite-volume formulation of the right hand side.

Direct inversion of a large block banded matrix becomes impractical in three dimensions because of the rapid increase of computational work and the large storage requirement. The LU-SGS scheme is one of the approximate factorization methods to alleviate the difficulties in three dimensions. Let subscripts f and s indicate fluid and species transport equations respectively. The loosely-coupled method solves the Navier-Stokes and species transport equations separately but the solutions are updated simultaneously at each time step.

$$LD^{-1}U\Delta Q = RHS \quad (4)$$

where

$$L_f D_f^{-1} U_f \Delta Q_f = RHS_f \quad (5)$$

$$\begin{aligned} L_f &= I + \alpha\Delta t(D_\xi^- A_f^+ + D_\eta^- B_f^+ + D_\zeta^- C_f^+ \\ &\qquad - A_f^- - B_f^- - C_f^-) \\ D_f &= I + \alpha\Delta t(A_f^+ + B_f^+ + C_f^+ - A_f^- - B_f^- - C_f^-) \\ U_f &= I + \alpha\Delta t(D_\xi^+ A_f^- + D_\eta^+ B_f^- + D_\zeta^+ C_f^- \\ &\qquad + A_f^+ + B_f^+ + C_f^+) \end{aligned} \quad (6)$$

$$L_s D_s^{-1} U_s \Delta Q_s = RHS_s \quad (7)$$

$$\begin{aligned} L_s &= I + \alpha\Delta t(D_\xi^- A_s^+ + D_\eta^- B_s^+ + D_\zeta^- C_s^+ \\ &\qquad - A_s^- - B_s^- - C_s^- - H) \\ D_s &= I + \alpha\Delta t(A_s^+ + B_s^+ + C_s^+ - A_s^- - B_s^- - C_s^-) \\ U_s &= I + \alpha\Delta t(D_\xi^+ A_s^- + D_\eta^+ B_s^- + D_\zeta^+ C_s^- \\ &\qquad + A_s^+ + B_s^+ + C_s^+) \end{aligned} \quad (8)$$

The loosely-coupled partially-implicit scheme includes the source Jacobian term H only in the L_s factor. Solving the equations in a loosely-coupled manner ignores such terms in the Jacobian matrix A, for example, as $\partial E_f / \partial Q_s$ and $\partial E_s / \partial Q_f$.

III. Parallelization methods

The original vector code ran inefficiently on cache-based systems. First, manual optimization that included array changes for better cache and Translation Lookaside Buffer utilization

enhanced the performance of the serial code by a factor of two on the Origin 3800 system. The parallelization on shared memory systems is easier than distributed memory machines because of the globally addressable space. The user does not have to worry about distributing and communicating data in separate address spaces.

The LU-SGS scheme in the code was vectorized on hyperplanes where $i+j+k=const$ where i,j, and k denote indices for three space dimensions. Figure 1 illustrates a two-dimensional example with $j+k=const$ hyperlines. Circles indicate grid cells to be solved while bullets are grid cells where the solutions are already updated. Calculations for the circles can be performed independently since the data for each circle have no dependencies. The key element was the conversion of three-dimensional indices (i,j,k) to two-dimensional ones $(ipoint, iplane)$[4]. Once the hyperplanes are formed, it is possible to parallelize the algorithm by partitioning the planes. The method has the limitation that parallelism is restricted to points within one hyperplane.

In order to improve memory access, the code has been converted manually to use a canonical ordering. The restructured code improves the serial performance by a factor of two, already a significant speed-up on its own. Then the processors are scheduled like a pipeline on the outermost loop level[12]. Sequential operations in each processor are performed in the cache. This approach exploits partial parallelism in loops that carry dependencies. Figure 2 illustrates an example for pipeline parallelization. A parallel region is placed around the outermost loop in k-dimension. The work is distributed in j-dimension. For each k, all threads execute a slice of j-index. The first processor starts from the lower-left corner and works on one slice of data for the first k-index. Other processors are waiting for data to be available. Once the first processor finishes its job, the second processor can start working on its slice for the same k-index. In the meantime, the first processor moves onto the next k-index. This process continues until all the processors become active. Then they all work concurrently to the opposite end. The efficiency of pipelining may be limited due to the wait in startup and finishing.

Both hyperplane and pipeline codes are parallelized using the Computer-Aided Parallelizer and Optimizer (CAPO)[13] parallelization tool for the OpenMP parallelization. The CAPO tool automates the insertion of compiler directives to facilitate parallel processing on shared memory machines. Due to the broad support of the OpenMP standard, the generated OpenMP codes can be run on a wide range of shared memory computers. The CAPO tool generates compiler directives in three stages: identification of parallel loops in the outer-most level, construction and optimization of parallel regions around parallel loops, and insertion of directives with a proper list of private, reduction and shared variables.

This task would have been very time consuming when performed manually, particularly in view of the fact that the code requires sophisticated parallelization techniques such as pipelined thread execution, which is not available via automatic parallelization of the vendor-supplied commercial compiler. The rapid tool based parallelization allows for the comparison of different strategies and to choose the most efficient implementation.

The parallelization is non-trivial, since the implementation gives rise to a number of conservative and actual data dependencies. The CAPO tool uses the extensive dependency analysis module of the ParaWise[14] system, and, based on the information resulting from the analysis, inserts OpenMP directives into the source code. The following features of the CAPO tool, which are not available via automatic compiler parallelization, are essential for the efficiency of the parallel code. The CAPO tool provides an extensive set of browsers to allow user interaction for improvements of the generated code. This makes it possible to interactively declare the scope of certain variables as either shared or private and thereby removing conservatively assumed dependencies, which would inhibit parallelization for the compiler. The CAPO tool optimizes the parallel code by merging the parallelized loops within a routine into a large parallel region. This reduces time spent in overhead to fork and join at the beginning and end of parallel loops.

IV. Results

The SGI Origin 3800 and Altix 3700 shared-memory systems are based on 0.6 GHz RISC and 1.5 GHz Intel Itanium-2 processors

respectively. Timings for the serial and the parallel executions were obtained using the -O2 optimization compiler flag during compilation.

In order to investigate the performance of parallel Gauss-Seidel methods for reacting flow, a scramjet problem has been used as a test case. The air-breathing rocket propulsion systems, which consume oxygen in the air, offer clear advantages by making vehicles lighter and more efficient. Fuel-air mixing and rapid combustion are of crucial importance for the success of scramjet engines since the spreading rate of the supersonic mixing layer decreases as the Mach number increases. In our test case, hydrogen fuel is injected transversely to incoming supersonic flow of air. The incoming air speed, pressure and temperature are assumed to be Mach 2, 1 atm and 1,000° K. Gaseous hydrogen is injected at the sonic speed through a hole at the bottom whose non-catalytic wall is cooled at 600° K. The length of combustion chamber is 40 times the diameter of injector. The Reynolds number based on the length is approximately 10^5. A 257 x 257 x 257 structured grid (approximately 17 million points) has been used with symmetric boundary conditions at the top and side walls. Supersonic flow boundary conditions are imposed at the inlet and outlet planes.

Figure 3 shows the speedup factors of the hyperplane method on an SGI Origin 3800 system. The code consists of Left Hand Side (LHS) and Right Hand Side (RHS) of the equation. The LHS represents the Gauss-Seidel algorithm while the RHS includes residual calculation routines. The parallel performance of the code worsens when the number of processors increases to 128. This problem appears to be due to complicated memory access of the hyperplane method as indicated by the LHS performance. Figure 4 shows the speedup factors of the pipeline method on the SGI Origin system. No performance degradation is observed since the pipeline improves memory access. However, the speedup factor of the RHS does not appear to increase as the number of processors increases from 64 to 128. Figure 5 shows clearly that the pipeline method outperforms the hyperplane method as the number of processors increases. Figure 6 shows relative speedup factors of the pipeline method over the hyperplane method. Even on single processor, the pipeline method is 4.58 times faster than the hyperplane method as shown by the LHS graph. Excluding the 128 processor case, the relative speedup for the LHS ranges from 4.58 to 6.08.

Figure 7 shows the speedup factors of the pipeline method on an SGI Altix 3700 system. It should be noted that the performance increases from 64 to 128 processors unlike with the Origin system in Fig. 4. Figure 8 compares the speedup factors for components of the RHS. The performance problems of residual calculation routines should be a subject of future investigations. Figure 9 shows the relative speedup of the Altix over the Origin. The relative speedup is the ratio of the Origin time and the Altix time on a given number of processors. It is not surprising that the Altix is two to three times faster than the Origin considering the speed of Altix chip is 2.5 times faster than the Origin's. What is interesting is that the best performance of the Altix seems to be at 128 processors. Finally, Figure 10 shows the effect of grid size on the speedup. The performance on the Altix improves as the grid size increases.

When compared to a 0.8 GHz Cray X1, the code on single processor of the Altix requires 1.54 times more time than a vectorized version on X1's single stream processor. However, the code on 4 Altix processors requires only 85 percent of the time on X1's 4 single stream processors (1 multi stream processor).

Summary

Parallelization methods have been studied for a symmetric Gauss-Seidel relaxation algorithm in conjunction with a loosely-coupled scheme for chemically reacting non-equilibrium flow. Both hyperplane and pipeline methods have been implemented into Real-Gas Aerodynamic Simulator code using OpenMP directives on cache coherent non-uniform memory access architecture. Performance of the parallelization methods have been investigated on SGI Altix and Origin shared memory systems. The pipeline method outperforms the hyperplane method partly because of the improved memory access. The Altix shows better performance than the Origin as the number of processors increases.

Acknowledgments

The authors thank H. Jin for helpful discussions.

References

1. Yoon, S. and Jameson, A., "Lower-Upper Symmetric Gauss-Seidel Method for the Euler and Navier-Stokes Equations," *AIAA Journal*, Vol. 26, Sept. 1988, pp. 1025-1026.
2. Shuen, J.S. and Yoon, S., "Numerical Study of Chemically Reacting Flows Using an LU-SSOR Scheme," *AIAA Journal*, Vol. 27, Dec. 1989, pp. 1752-1760.
3. Park, C. and Yoon, S., "Calculation of Real-Gas Effects on Blunt-Body Trim Angles," *AIAA Journal*, Vol. 30, Apr. 1992, pp. 999-1007.
4. Yoon, S. and Kwak, D., "Three-Dimensional Incompressible Navier-Stokes Solver using Lower-Upper Symmetric Gauss-Seidel Algorithm," *AIAA Journal*, Vol. 29, June 1991, pp. 874-875.
5. Yoon, S. and Kwak, D., "Multigrid Convergence of an Implicit Symmetric Relaxation Scheme," *AIAA Journal*, Vol. 32, May 1994, pp. 950-955.
6. Chen, C.L., McCroskey, W.J., and Obayashi, S., "Numerical Solutions of Forward-Flight Rotor Flow using an Upwind Method," AIAA Paper 89-1846, June 1989.
7. Soetrisno, M., Imlay, S.T., and Roberts, D.W., and Taflin, D.E., "Development of a 3-D Zonal Implicit Procedure for Hybrid Structured-Unstructured Grids," AIAA Paper 96-0167, Jan. 1996.
8. Sharov, D. and Nakahashi, K., "Reordering of 3-D Hybrid Unstructured Grids for Vectorized LU-SGS Navier-Stokes Computations," AIAA Paper 97-2102, June 1997.
9. Yoon, S., "Calculation of Supersonic Combustion using Implicit Schemes," *AIAA Journal*, Vol. 42, Dec. 2004, pp. 2482-2489.
10. Bailey, D., Barton, J., Lasinski, T., and Simon, H., "The NAS Parallel Benchmarks," NAS TR-91-002, Jan. 1991.
11. Van der Wijngaart, R., Sarukkai, S., and Mehra, P., "Analysis and Optimization of Software Pipeline Performance on MIMD Parallel Computers," NAS TR-97-003, Feb. 1997.
12. Jin, H., Frumkin, M., and Yan, J., "The OpenMP Implementation of NAS Parallel Benchmarks and its Performance," NAS TR-99-011, Oct. 1999.
13. Jin, H., Frumkin, M., and Yan, J., "Code parallelization with CAPO," NAS TR-01-008, Aug. 2001.
14. http://www.parallelsp.com/parawise.htm.

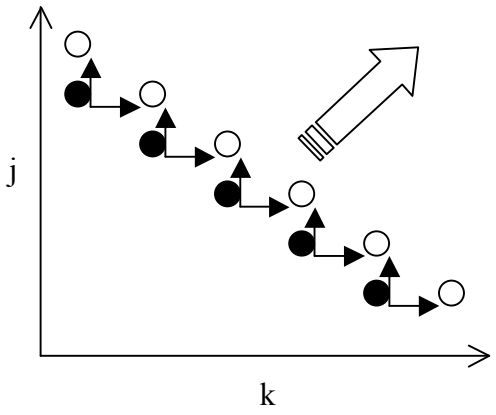

Fig. 1. Hyperplane parallelization: Shown is a $j+k=const$ hyperline for two dimensions ($i+j+k=const$ for three dimensions).

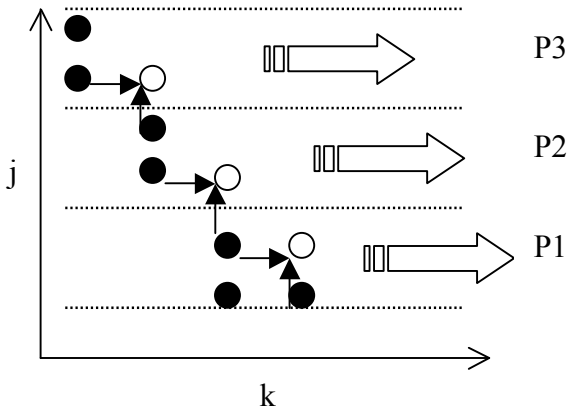

Fig. 2. Pipeline parallelization: Processors are scheduled along k-direction. Next processor waits for updated data from previous processor for a given k-index.

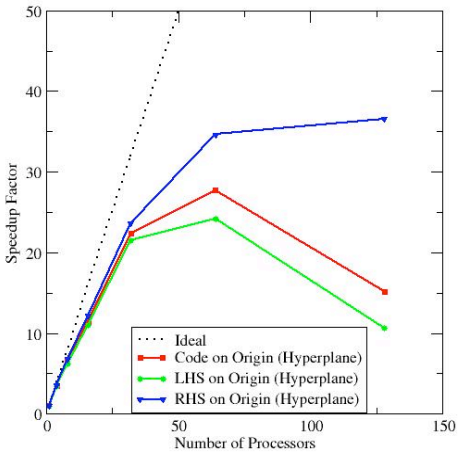

Fig. 3. Speedup factors of the hyperplane method on SGI Origin 3800 sytem

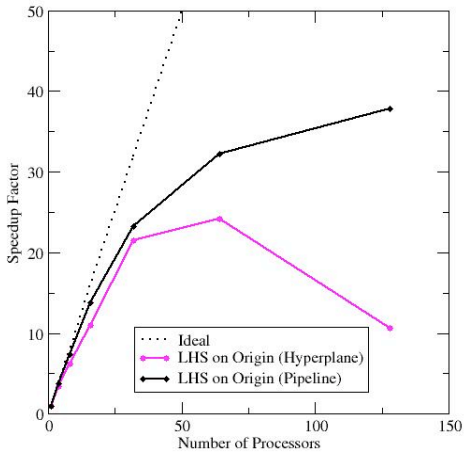

Fig. 5. Comparison of hyperplane and pipeline methods for the Left Hand Side (Gauss-Seidel algorithm)

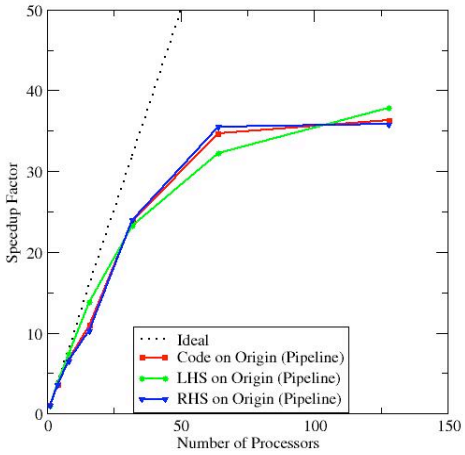

Fig. 4. Speedup factors of the pipeline method on SGI Origin 3800 system

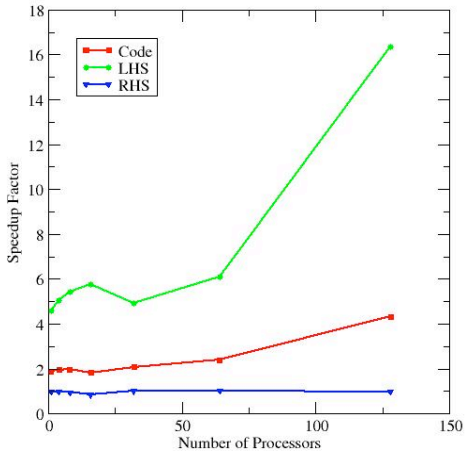

Fig. 6. Relative speedup factors of the pipeline method over the hyperplane method

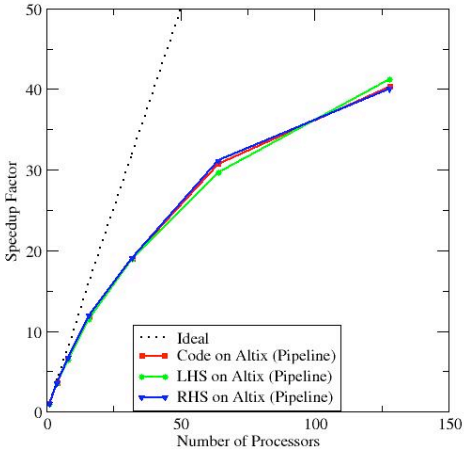

Fig. 7. Speedup factors of the pipeline method on SGI Altix 3700 system

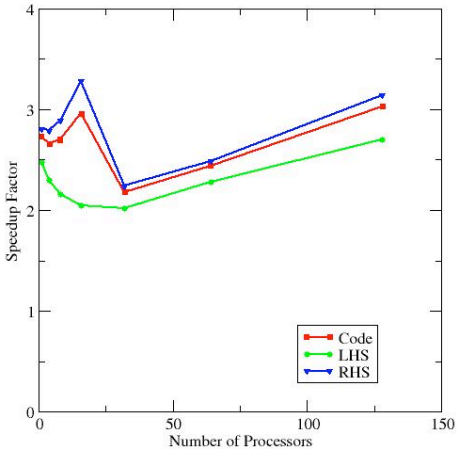

Fig. 9. Relative speedup factors of SGI Altix 3700 over SGI Origin 3800

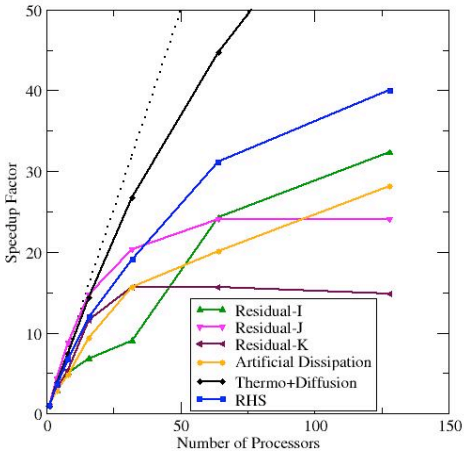

Fig. 8. Speedup factors for components of the Right Hand Side (Unrelated to Gauss-Seidel algorithm)

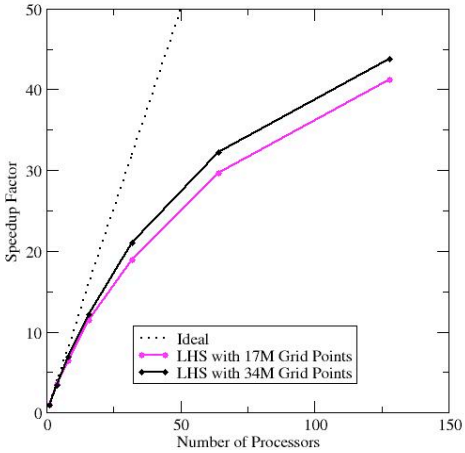

Fig. 10. Effect of grid size on parallel performance